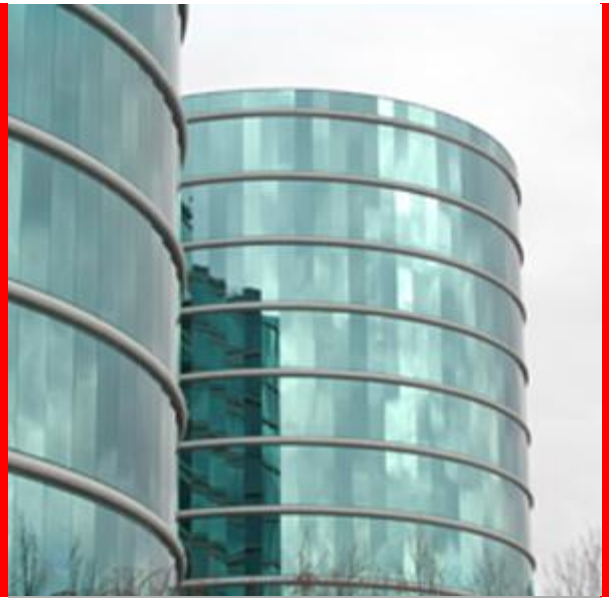

ORACLE®



ORACLE®

Keine Angst vor SQL Injection oder "Wie man Datenbankzugriffe sicher implementiert"

Carsten Czarski
Oracle Deutschland B.V. & Co KG

SQL Injection: Was ist das...?

- Ein böswilliger Angreifer
  - ... nutzt eine Schwachstelle
  - ... in einer Applikation
  - ... um SQL-Kommandos einzuschleusen
  - ... und so Daten zu manipulieren oder auszuspähen

SQL Injection:
Das "klassische" Beispiel

- Daten ausspähen durch Manipulieren der URL

DEPTNO: DEPT 10

Printing all Employees in DEPTNO 20 or 1=1

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	1980-12-17 00:00:00.0	12800	null	20
7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.0	25600	300	30
7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.0	20000	500	30
7566	JONES	MANAGER	7839	1981-04-02 00:00:00.0	47600	null	20
7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.0	20000	1400	30
7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.0	45600	null	30
7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.0	39200	null	10
7788	SCOTT	ANALYST	7566	1987-04-19 00:00:00.0	48000	null	20
7839	KING	PRESIDENT	null	1981-11-17 00:00:00.0	80000	null	10
7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.0	24000	0	30
7876	ADAMS	CLERK	7788	1987-05-23 00:00:00.0	17600	null	20
7900	JAMES	CLERK	7698	1981-12-03 00:00:00.0	15200	null	30
7902	FORD	ANALYST	7566	1981-12-03 00:00:00.0	48000	null	20
7934	MILLER	CLERK	7782	1982-01-23 00:00:00.0	20800	null	10

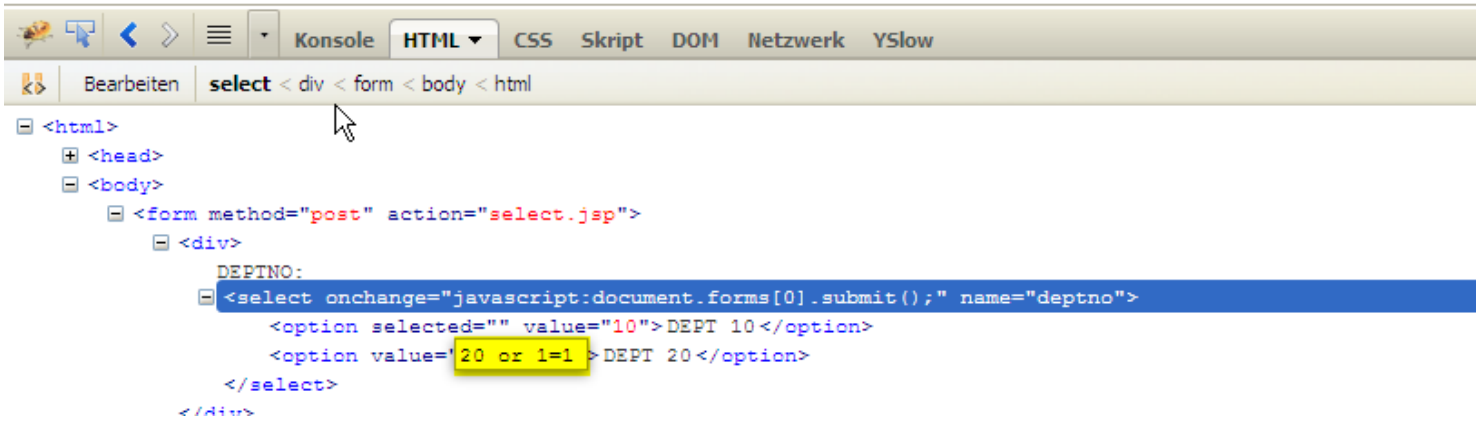
SQL Injection:
Das "klassische" Beispiel

- Auch HTTP POST ist kein Hindernis

DEPTNO: DEPT 10

Printing all Employees in DEPTNO 10

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.0	39200	null	10
7839	KING	PRESIDENT	null	1981-11-17 00:00:00.0	80000	null	10
7934	MILLER	CLERK	7782	1982-01-23 00:00:00.0	20800	null	10



Hintergrund

- Eingaben des Benutzers werden *unkontrolliert* ins SQL integriert = SQL Injection-Lücke

```
<%
  Class.forName("oracle.jdbc.OracleDriver");
  Connection con = DriverManager.getConnection (
   "jdbc:oracle:thin:@sccloud030:1521:orcl", "scott", "tiger"
  );
  PreparedStatement pstmt = con.prepareStatement (
    "select * from emp " +
    " where deptno = "+request.getParameter("deptno") + " and sal is not null"
  );
  ResultSet rs = pstmt.executeQuery();

%>
<div>
```

Erste Ideen ...

- Prüfen der Eingabe auf bestimmte Schlüsselworte
  `--, 1=1, or, and, select, ...`

- Aber:
  - Kann oft umgangen werden
  - Vollständige Überprüfung *aller* Schlüsselworte schwierig
  - Bei Freitexteingaben ("Bemerkungen") oft gar nicht möglich

- **Generische Lösung ist gefragt!**

Lösungsansatz: Bind-Variablen

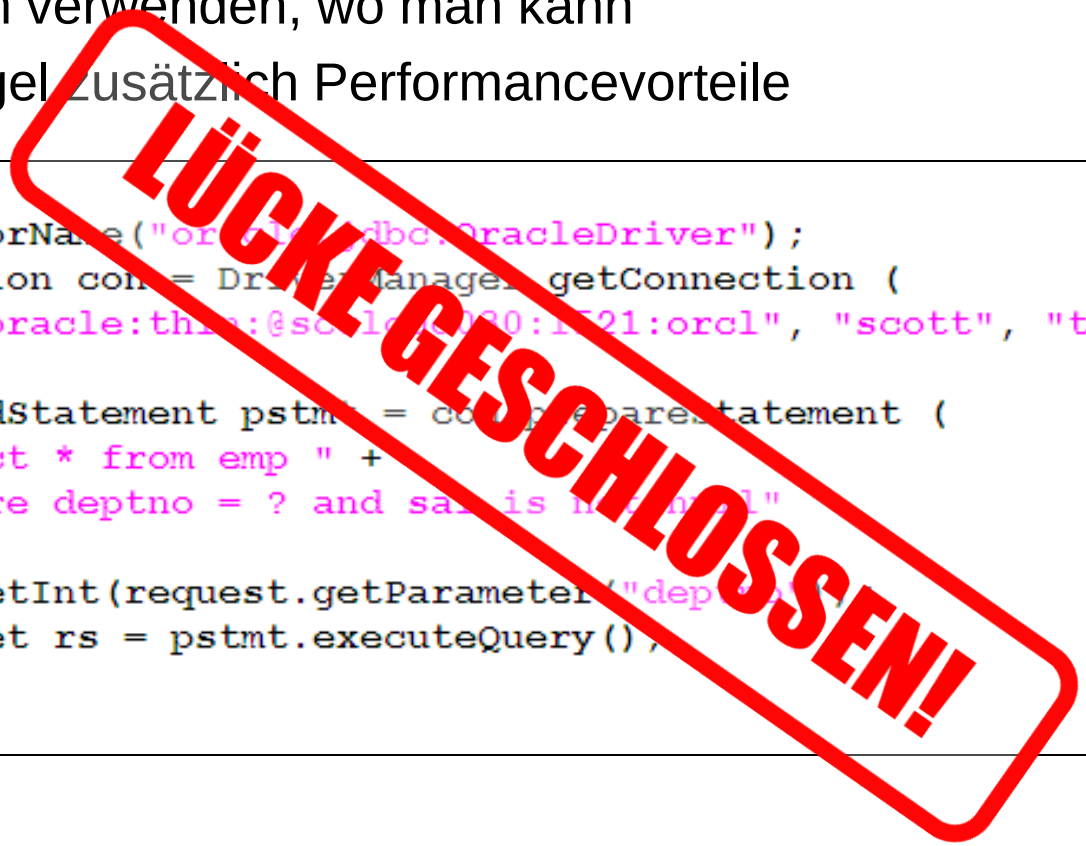
- Java: PreparedStatement oder CallableStatement
  - Sollte man verwenden, ***wo man kann***
  - In der Regel zusätzlich Performancevorteile

```
<%
  Class.forName("oracle.jdbc.OracleDriver");
  Connection con = DriverManager.getConnection (
   "jdbc:oracle:thin:@sccloud030:1521:orcl", "scott", "tiger"
  );
  PreparedStatement pstmt = con.prepareStatement (
    "select * from emp " +
    " where deptno = ? and sal is not null"
  );
  pstmt.setInt(request.getParameter("deptno"));
  ResultSet rs = pstmt.executeQuery();

%>
```

Lösungsansatz: Bind-Variablen

- Java: PreparedStatement oder CallableStatement
  - Sollte man verwenden, wo man kann
  - In der Regel zusätzlich Performancevorteile

```
<%
  Class.forName("oracle.jdbc.OracleDriver");
  Connection con = DriverManager.getConnection (
   "jdbc:oracle:thin:@scl[illegible]:1521:orcl", "scott", "tiger"
  );
  PreparedStatement pstmt = con.prepareStatement (
    "select * from emp " +
    " where deptno = ? and sal is not null"
  );
  pstmt.setInt(request.getParameter("deptno"));
  ResultSet rs = pstmt.executeQuery();

%>
```



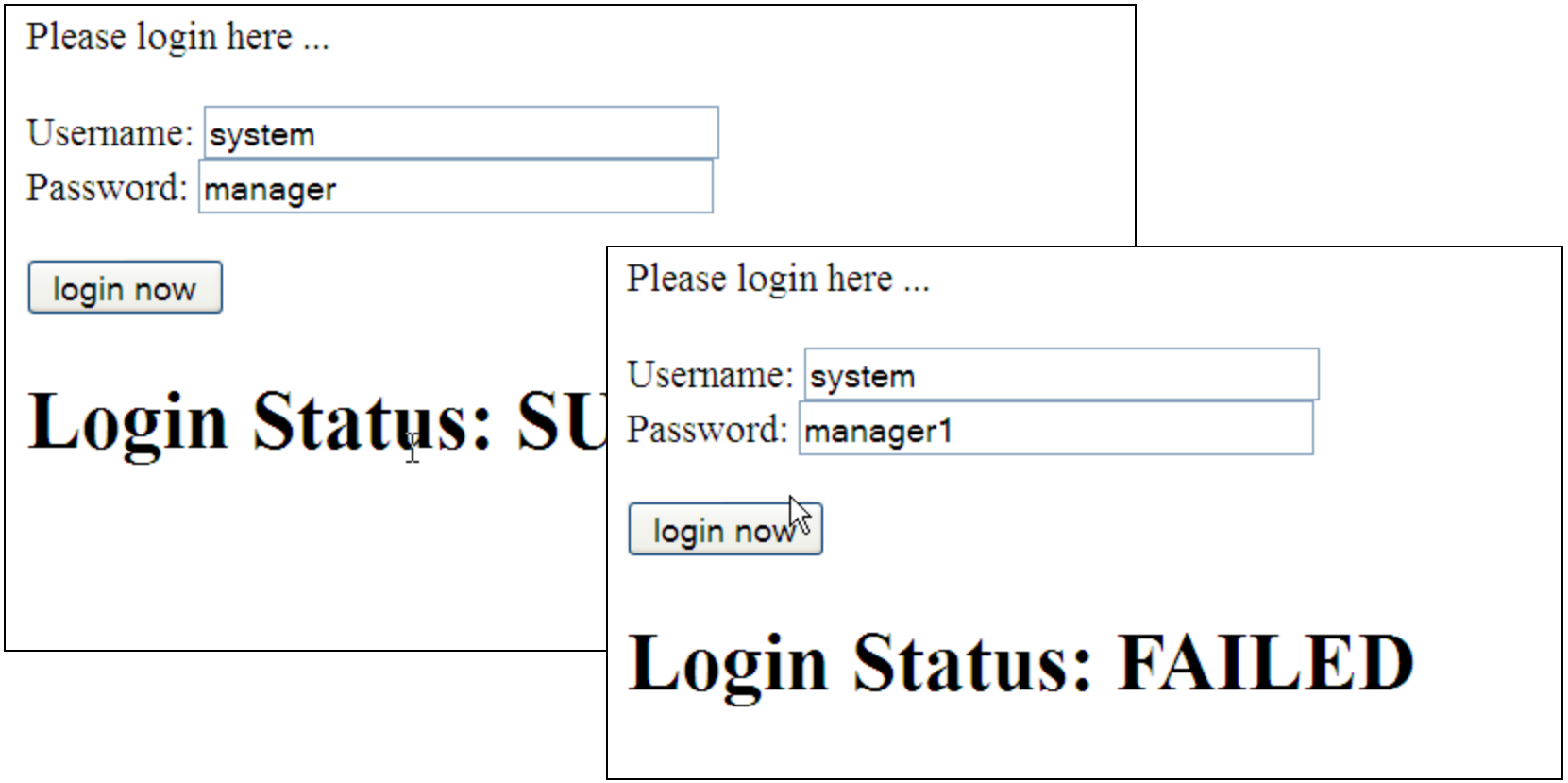
SQL Injection
Weiter denken ...

- SQL Injection im Zusammenspiel mit Stored Procedures

```
<%
  Connection con = null;
  CallableStatement cstmt = null;
  String sLoginStatus = "NOT ATTEMPTED";
  if (request.getParameter("username") != null) {
    try {
     Class.forName("oracle.jdbc.OracleDriver");
     con = DriverManager.getConnection ("jdbc:oracle:thin:@sccloud030:1521:orcl", "scott", "tiger");

     cstmt = con.prepareCall("{? = call do_login(?,?)}");
     cstmt.setString(2, request.getParameter("username"));
     cstmt.setString(3, request.getParameter("password"));
     cstmt.registerOutParameter(1, java.sql.Types.VARCHAR);
     cstmt.execute();
     sLoginStatus = cstmt.getString(1);
     cstmt.close();
     con.close();
    } catch (Exception e) {
     sLoginStatus = e.getMessage();
  } finally {
     cstmt.close();
     con.close();
    }
  }
%>
```

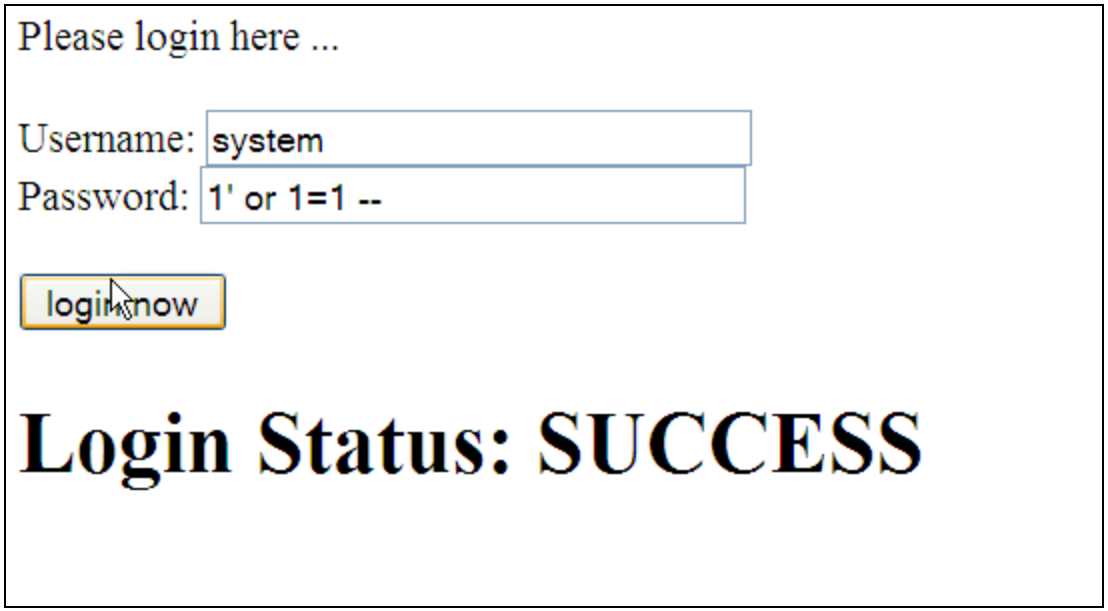
Probieren wir es aus!

- **SYSTEM/manager** ist gültig



Probieren wir es aus!

- Aber es geht auch anders ...



... wie kann das sein?

SQL in Stored Procedures

- Eine SQL Injection-Lücke: In einer Stored Procedure!

```
create or replace function do_login(
 p_username in varchar2,
 p_password in varchar2
) return varchar2 is
  v_success number;
begin
  execute immediate
   'select count(*) from login_users ' ||
   'where username = '''||p_username||''' and password = '''||p_password||''''
  into v_success;

  if v_success = 0 then
    return 'FAILED';
  else
    return 'SUCCESS';
  end if;
end;
/
sho err
```

Fehlermeldungen ...

Können der Freund des Hackers sein

Please login here ...

Username: system

```
1' or 1=dbms_metadata.open(null,(select
listagg(username,'<br>') within group (order by username)
from  all_users)) --
```

Password:

login now

Login Status: ORA-31600: invalid input value ANONYMOUS
APEXWS_110
APEXWS_12
APEXWS_13
APEXWS_30
APEXWS_50
APEXWS_70
APEXWS_90
APEX_040000
APEX_040100

Lösungsansatz: Bind Variablen

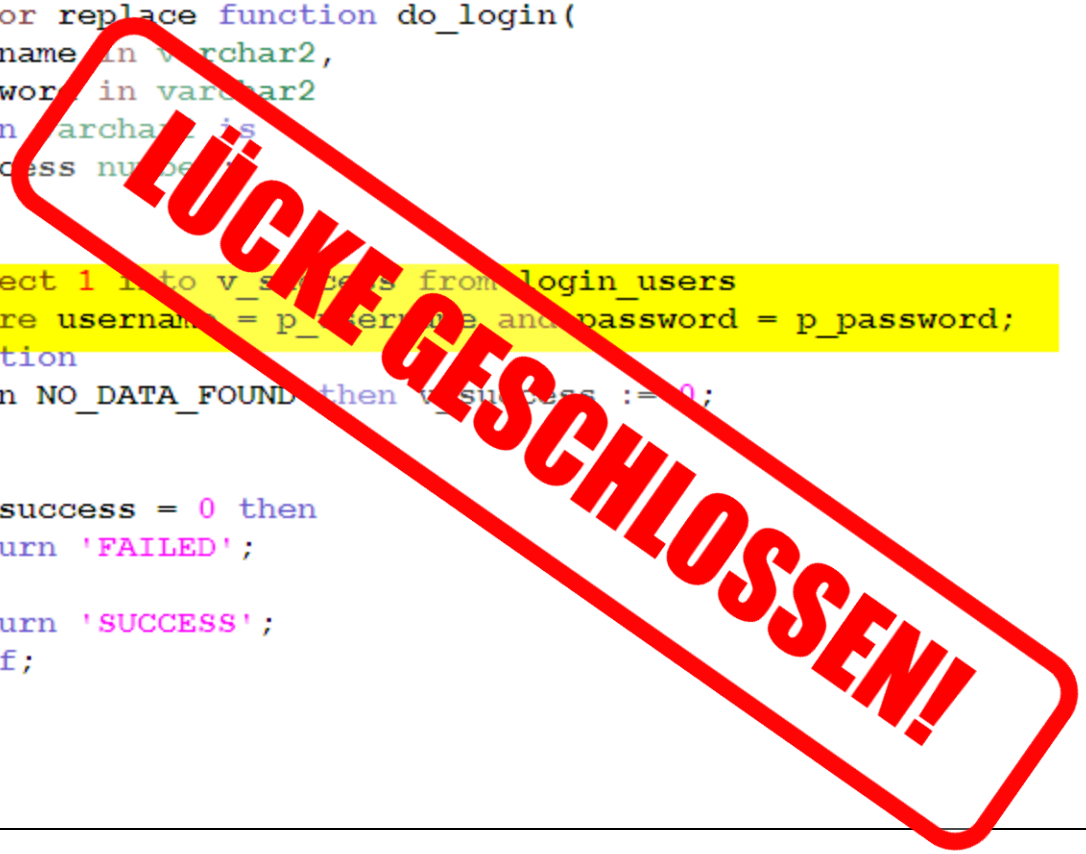
- In PL/SQL sind Bindvariablen der Default

```
create or replace function do_login(
 p_username in varchar2,
 p_password in varchar2
) return varchar2 is
  v_success number;
begin
  begin
    select 1 into v_success from login_users
    where username = p_username and password = p_password;
  exception
    when NO_DATA_FOUND then v_success := 0;
  end;

  if v_success = 0 then
    return 'FAILED';
  else
    return 'SUCCESS';
  end if;
end;
/
sho err
```

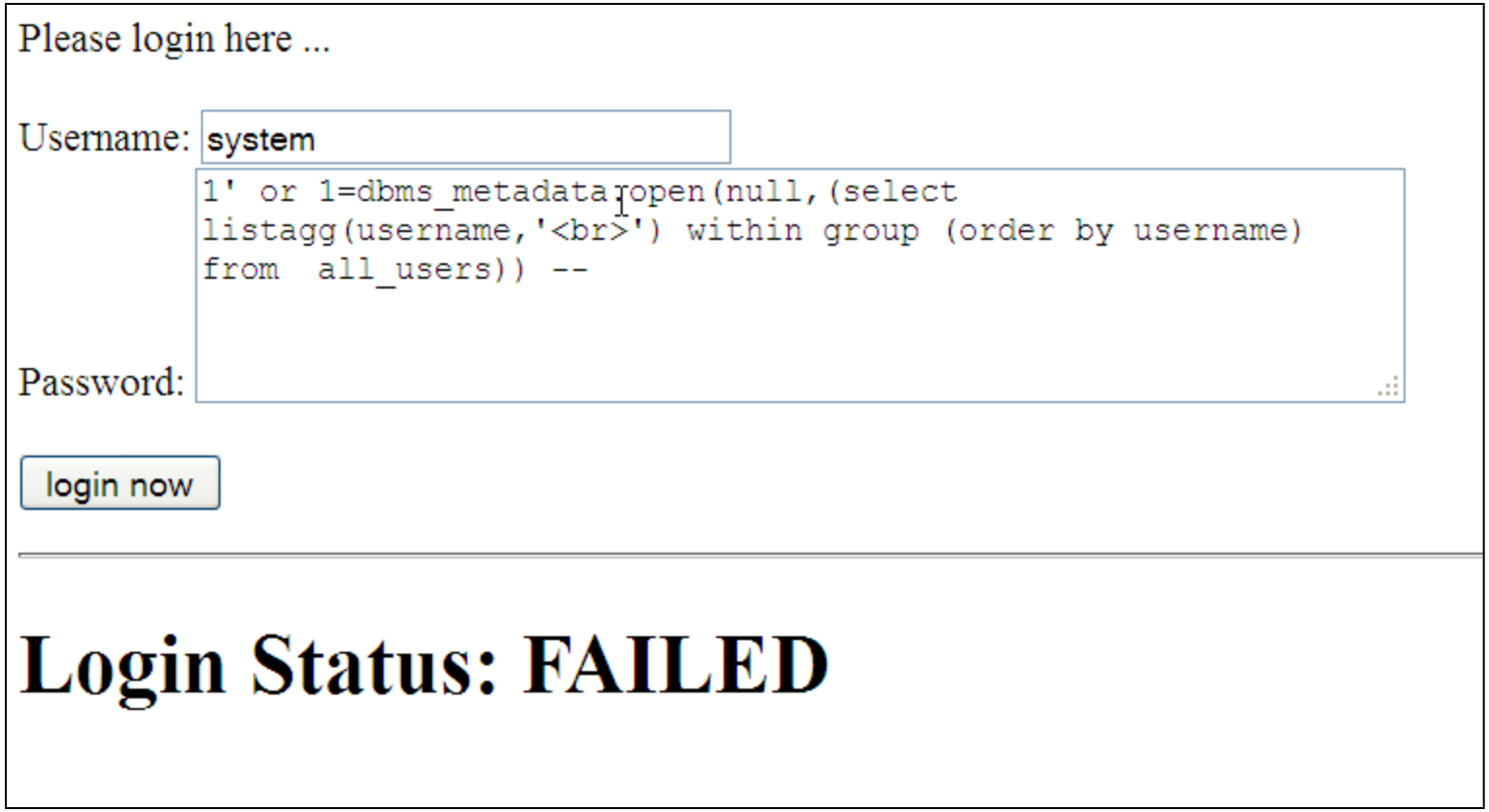
Lösungsansatz: Bind Variablen

- In PL/SQL sind Bindvariablen der Default

```
create or replace function do_login(
 p_username in varchar2,
 p_password in varchar2
) return varchar2 is
  v_success number;
begin
  begin
    select 1 into v_success from login_users
    where username = p_username and password = p_password;
  exception
    when NO_DATA_FOUND then v_success := 0;
  end;

  if v_success = 0 then
    return 'FAILED';
  else
    return 'SUCCESS';
  end if;
end;
/
sho err
```

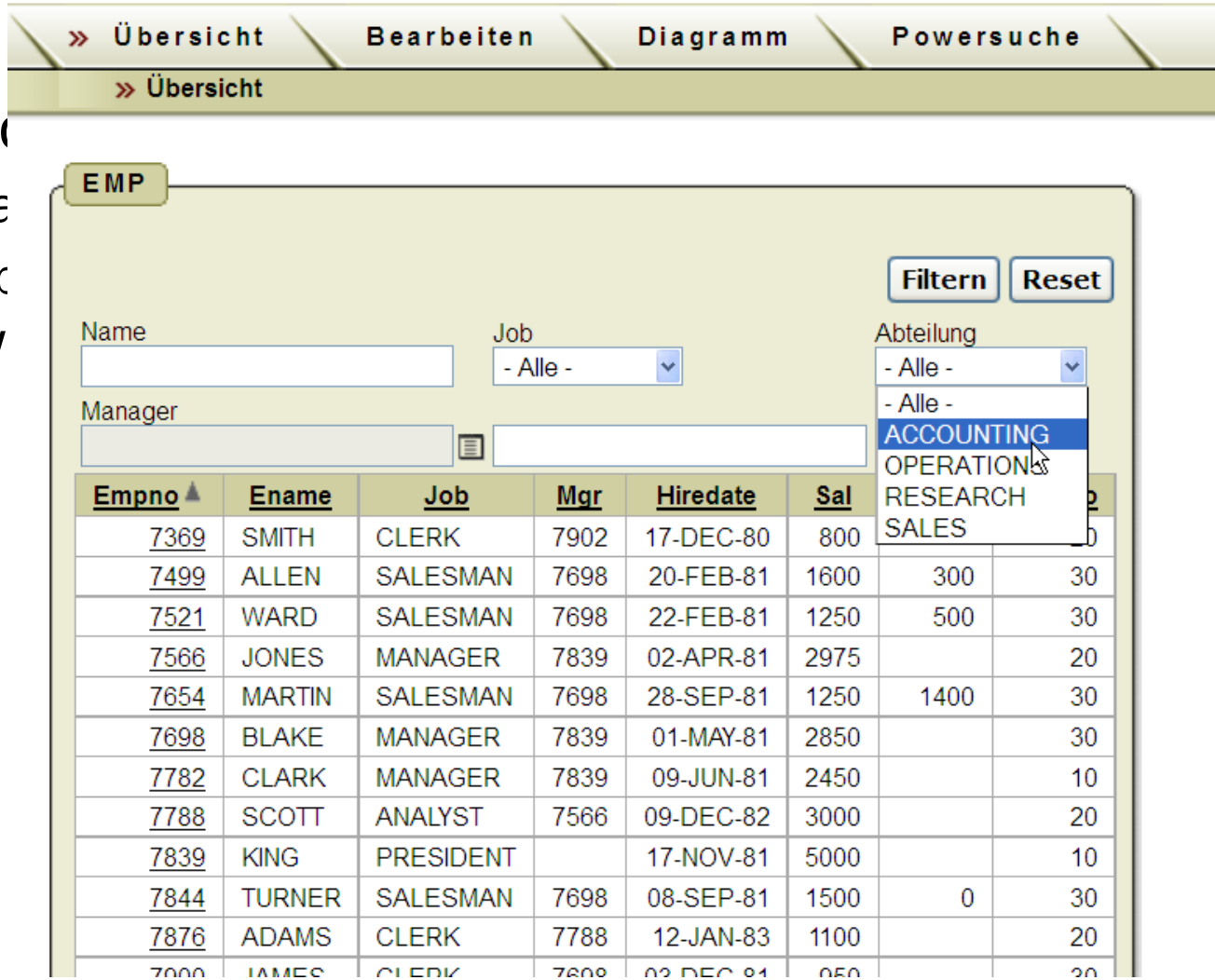


Der Angriff funktioniert nicht mehr ...



Bind-Variablen: Grenzen

- SQL-Struktur s
  - Dynamische Ta
  - Dynamische Sp
  - Dynamische W
- Bindevariablen



Dynamisches SQL in einer JSP

- Filter im SQL richtet sich nach JSP-Parametern

```
<%
  Class.forName("oracle.jdbc.OracleDriver");
  Connection con = DriverManager.getConnection (
   "jdbc:oracle:thin:@sccloud030:1521:orcl", "scott", "tiger"
  );
  String sSql = "select * from emp where ";
  // wenn "job" gegeben, dann Filter auf JOB
  if (request.getParameter("job") != null) {
    sSql = sSql + " job = '" + request.getParameter("job")+"' and ";
  }
  // wenn "sal" gegeben, dann Filter auf SAL - Vergleich auch dynamisch
  if (request.getParameter("sal") != null) {
    sSql = sSql + " sal " + request.getParameter("sal-vergleich") +
                        " " + request.getParameter("sal") + " and ";
  }
  sSql = sSql + " 1=1 ";
  PreparedStatement pstmt = con.prepareStatement (sSql);
  ResultSet rs = pstmt.executeQuery();

%>
```

Muss es wirklich verkettet werden ...?

Optimizer eliminiert nicht benötigte Zweige

```
<%
  Class.forName("oracle.jdbc.OracleDriver");
  Connection con = DriverManager.getConnection (
   "jdbc:oracle:thin:@sccloud030:1521:orcl", "scott", "tiger"
  );
  PreparedStatement pstmt = con.prepareStatement (
    "with parameters as ( " +
    "  select ? as jobfilter, ? as salfilter, ? as salvergleich from dual " +
    ") " +
    "select * from emp e, parameters p " +
    "where (job = p.jobfilter or p.jobfilter is null) " +
    "and (" +
    " (sal >  p.salfilter and p.salvergleich =  '>') or "+
    " (sal <  p.salfilter and p.salvergleich =  '<') or "+
    " (sal =  p.salfilter and p.salvergleich =  '=') or "+
    " (sal >= p.salfilter and p.salvergleich = '>=') or "+
    " (sal <= p.salfilter and p.salvergleich = '<=') "+
    ")"
  );
  pstmt.setString(1, request.getParameter("job");
  pstmt.setInt(2, request.getParameter("sal");
  pstmt.setString(3, request.getParameter("sal-vergleich");
  ResultSet rs = pstmt.executeQuery();
%>
```

Geht es nicht doch statisch ...?

Der Optimizer eliminiert nicht benötigte Zweige

```
<%
  Class.forName("oracle.jdbc.OracleDriver");
  Connection con = DriverManager.getConnection (
   "jdbc:ora  :thin:@sccloud030:1521:orcl", "scott", "tiger"
  );
  Prepared tatement pstmt = con.prepareStatement (
    "wit  para  t  a  ( " +
    "  select  jobfi  er, ? as salfilter, ? as salvergleich vom dual " +
    ") " +
    "select  from  ,parameters p " +
    "where (job = p.jo  ter or  jobfilter is null) " +
    "and (" +
    " (sal >  p.salfilter a  lvergleich =  '>') or "+
    " (sal <  p.salfilter and  rgleich =  '<') or "+
    " (sal =  p.salfilter and p.  leich =  '=') or "+
    " (sal >= p.salfilter and p.sal  ch =  '>=') or "+
    " (sal <= p.salfilter and p.salver  =  '<=') "+
    ")"
  );
  pstmt.setInt(1, request.getParameter("job"));
  pstmt.setInt(2, request.getParameter("sal");
  pstmt.setInt(3, request.getParameter("sal-vergleich");
  ResultSet rs = pstmt.executeQuery();

%>
```

LÜCKE GESCHLOSSEN!

Und das stimmt wirklich ...?

- Ein Beispiel ...

```
with parameters as (
  select 'CLERK' as jobfilter, 1000 as salfilter, '>' as salvergleich from dual
)
select * from parameters p, emp e
```

```
Ausführungsplan
----------------------------------------------------------

-------------------------------------------------------------------------------
| Id  | Operation          | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |        |   137K|  5241K|  1142   (2)| 00:00:14 |
|   1 |  NESTED LOOPS      |        |   137K|  5241K|  1142   (2)| 00:00:14 |
|   2 |   FAST DUAL        |        |     1 |       |     2   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| BIGEMP |   137K|  5241K|  1140   (2)| 00:00:14 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter("JOB"='CLERK' AND "SAL">1000)
```

Lösungsansatz: "Mißtrauen"

- "*Misstraue allem, was vom Browser kommt*"
  - URL-Parameter
  - Cookie-Inhalte(auch die lassen sich manipulieren)
  - Inhalte aus Web-Formularen

- Vor Verwendung prüfen oder besser noch: maskieren ...
  - Für ***Literale*** in der WHERE-Klausel ...
    - Einbau in einfachen Anführungszeichen: 'CLERK' bzw. '10'
    - In Nutzereingabe enthaltene ' dann als ' ' *verdoppeln*
  - Für ***Objektnamen*** (Tabelle, Spalte, Prozedur, ...)
    - In doppelte Anführungszeichen (") einschließen
    - Doppelte Anführungszeichen in Nutzereingabe verbieten!

Eingaben maskieren ...
Der einfachste Ansatz ist der Beste

- Einfache Anführungszeichen (') maskieren

```
<%
  Class.forName("oracle.jdbc.OracleDriver");
  Connection con = DriverManager.getConnection (
    "jdbc:oracle:thin:@sccloud030:1521:orcl", "scott", "tiger"
  );
  PreparedStatement pstmt = con.prepareStatement (
    "select * from emp " +
    "where to_char(deptno) = '" +
       request.getParameter("deptno").replaceAll("'","''") +
    "' and sal is not null");
  ResultSet rs = pstmt.executeQuery();
%>
```

Eingabe	Im SQL ...
10	to_char(deptno) = '10'
10' or 1=1 --	to_char(deptno) = '10'' or 1=1 --'
10'' or 1=1 --	to_char(deptno) = '10'''' or 1=1 --'

In Stored Procedures: DBMS_ASSERT

***to assert something* = etwas sicherstellen**

- Stellt sicher, dass ein SQL-Parameter keine SQL Injection-Gefahr darstellt
  - Nutzung **vor** dem "Einbau" in das SQL-Kommando
  - Bei SQL Injection-Gefahr: Fehlermeldung

Summary of DBMS_ASSERT Subprograms

Table 26-1 DBMS_APPLICATION_INFO Package Subprograms

Subprogram	Description
ENQUOTE_LITERAL Function	Enquotes a string literal
ENQUOTE_NAME Function	Encloses a name in double quotes
NOOP Functions	Returns the value without any checking
QUALIFIED_SQL_NAME Function	Verifies that the input string is a qualified SQL name
SCHEMA_NAME Function	Verifies that the input string is an existing schema n
SIMPLE_SQL_NAME Function	Verifies that the input string is a simple SQL name
SQL_OBJECT_NAME Function	Verifies that the input parameter string is a qualifiec

PL/SQL Paket DBMS_ASSERT

Anwendung für Literale ...

```
SQL> select dbms_assert.enquote_literal(q'#King#') from dual;

DBMS_ASSERT.ENQUOTE_LITERAL(Q'#KING#')
--------------------------------------------------------------------------------
'King'

1 Zeile wurde ausgewählt.


SQL> select dbms_assert.enquote_literal(q'#King' or 1=1 --#') from dual;
select dbms_assert.enquote_literal(q'#King' or 1=1 --#') from dual
       *
FEHLER in Zeile 1:
ORA-06502: PL/SQL: numerischer oder Wertefehler
ORA-06512: in "SYS.DBMS_ASSERT", Zeile 342
ORA-06512: in "SYS.DBMS_ASSERT", Zeile 411
```

PL/SQL Paket DBMS_ASSERT

Anwendung für Literale ...

```
SQL> select dbms_assert.enquote_literal(q'#King' or 1=1 --#') from dual;
select dbms_assert.enquote_literal(q'#King' or 1=1 --#') from dual
       *
FEHLER in Zeile 1:
ORA-06502: PL/SQL: numerischer oder Wertefehler
ORA-06512: in "SYS.DBMS_ASSERT", Zeile 342
ORA-06512: in "SYS.DBMS_ASSERT", Zeile 411


SQL> select dbms_assert.enquote_literal(q'#King'' or 1=1 --#') from
dual;

DBMS_ASSERT.ENQUOTE_LITERAL(Q'#KING''OR1=1--#')
--------------------------------------------------------------------
'King'' or 1=1 --'

1 Zeile wurde ausgewählt.
```

PL/SQL Paket DBMS_ASSERT

Anwendung für Objektnamen ...

```
SQL> select dbms_assert.QUALIFIED_SQL_NAME('EMP') from dual;

DBMS_ASSERT.QUALIFIED_SQL_NAME('EMP')
-------------------------------------------------------------------
EMP

1 Zeile wurde ausgewählt.

SQL> select dbms_assert.QUALIFIED_SQL_NAME('"EMP"') from dual;

DBMS_ASSERT.QUALIFIED_SQL_NAME('"EMP"')
-------------------------------------------------------------------
"EMP"

1 Zeile wurde ausgewählt.
```

PL/SQL Paket DBMS_ASSERT

Anwendung für Objektnamen ...

```
SQL> select dbms_assert.QUALIFIED_SQL_NAME('EMP"') from dual;
select dbms_assert.QUALIFIED_SQL_NAME('EMP"') from dual
       *

FEHLER in Zeile 1:
ORA-44004: Ungültiger qualifizierter SQL-Name
ORA-06512: in "SYS.DBMS_ASSERT", Zeile 207
```

- Verschiedene Funktionen verfügbar
  - ENQUOTE_LITERAL für Literale in WHERE-Klausel
  - ENQUOTE_NAME für Objektnamen
  - Weitere Funktionen
    SIMPLE_SQL_NAME, QUALIFIED_SQL_NAME,
    SQL_OBJECT_NAME, SCHEMA_NAME

Fazit: SQL Injection

- Aufgabe des Anwendungsentwicklers
- Einfache "Faustregeln"
  - Bind-Variablen verwenden
  - Parameter prüfen (PL/SQL: DBMS_ASSERT)
  - Code-Review (insbesondere für dynamisches SQL)

Für Stored Procedures gilt:

Wo EXECUTE IMMEDIATE, OPEN_FOR oder DBMS_SQL sind ...

... muss auch DBMS_ASSERT sein.

Carsten.Czarski@oracle.com

http://tinyurl.com/apexcommunity

http://sql-plsql-de.blogspot.com
http://oracle-text-de.blogspot.com
http://oracle-spatial.blogspot.com

http://plsqlexecoscomm.sourceforge.net
http://plsqlmailclient.sourceforge.net

Twitter: @cczarski @oraclebudb